Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

Технології розроблення програмного забезпечення

Тема: Взаємодія компонентів системи.

Виконав
студент групи ІА-34:
Жительний В.В.

Перевірив:
Мягкий М.Ю.

Київ 2025

Мета: Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service oriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

Тема:

21. **Online radio station** (iterator, adapter, factory method, facade, visitor, client-server)

Додаток повинен служити сервером для радіостанції з можливістю мовлення на радіостанцію (64, 92, 128, 196, 224 kb/s) в потоковому режимі; вести облік підключених користувачів і статистику відвідувань і прослуховувань; налаштувати папки з піснями і можливість вести списки програвання або playlists (не відтворювати всі пісні).

Хід роботи

Систему спроєктовано на базі клієнт-серверної архітектури з використанням фреймворку Spring Boot.

- Серверна частина: Реалізована на мові Java. Вона відповідає за обробку бізнес-логіки, керування плейлистами, трансляцію аудіопотоку та взаємодію з базою даних. Сервер надає доступ до функціоналу через REST API та MVC-контролери.
- Клієнтська частина: Реалізована у вигляді веб-сторінок, які відображаються у браузері користувача. Браузер виступає як «тонкий клієнт», відправляючи HTTP-запити на сервер для отримання контенту та керування станціями.
- Middleware / Протокол взаємодії: Взаємодія між клієнтом та сервером відбувається через протокол HTTP.
    - RadioController реалізує REST-підхід, повертаючи дані у форматі JSON (наприклад, список станцій) або медіа-потік (HLS).
    - WebPageController та AdminController використовують підхід MVC (Model-View-Controller) для генерації HTML-сторінок на стороні сервера.

Реалізація компонентів системи

Клас RadioController надає кінцеві точки для отримання списку станцій та аудіопотоку. Він використовує RadioFacade для отримання даних, приховуючи деталі реалізації.

```
@RestController
public class RadioController {

    private final RadioFacade radioFacade;

    @Autowired private ConnectionLogRepository connectionLogRepository;
    @Autowired private StationRepository stationRepository;

    @Autowired
    public RadioController(RadioFacade radioFacade) {
```

```
        this.radioFacade = radioFacade;
    }

    @GetMapping("/stations")
    public List<Station> getAllStations() {
        return radioFacade.getAllStations();
    }

    @GetMapping("/hls/{stationId}/stream.m3u8")
    public ResponseEntity<Resource> getHlsManifest(@PathVariable String
stationId, HttpServletRequest request) {
        Path path = Paths.get(System.getProperty("user.home"), "radio-hls-
static", "hls", stationId, "stream.m3u8");
        Resource resource = new FileSystemResource(path);

        try {
            Station station =
stationRepository.findById(Long.parseLong(stationId)).orElse(null);

            if (station != null) {
                String ip = request.getRemoteAddr(); // IP адреса
                String userAgent = request.getHeader("User-Agent"); // Браузер

                ConnectionLog log = new ConnectionLog(station, ip, userAgent);
                connectionLogRepository.save(log);

            }
        } catch (Exception e) {
            System.err.println("Stats error: " + e.getMessage());
        }

        if (!resource.exists()) {
            return ResponseEntity.notFound().build();
        }

        return ResponseEntity.ok()
                .cacheControl(CacheControl.noCache())

.contentType(MediaType.parseMediaType("application/vnd.apple.mpegurl"))
                .body(resource);
    }

    @PostMapping("/api/v1/favorites")
    public ResponseEntity<String> addToFavorites(
            @RequestParam Long stationId,
            @AuthenticationPrincipal UserDetails userDetails
    ) {
        if (userDetails == null) {
            return ResponseEntity.status(401).body("You must contain login
first");
        }

        try {
            String message =
radioFacade.addStationToFavorites(userDetails.getUsername(), stationId);

            return ResponseEntity.ok(message);

        } catch (Exception e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        }
    }

    @GetMapping("/api/v1/station/{id}/report")
    public String getStationReport(@PathVariable Long id) {
        return radioFacade.getStationReport(id);
```

```
    }
}
```

Клас AdminController відповідає за керування контентом.

```
@Controller
@RequestMapping("/admin")
public class AdminController {

    @Autowired private RadioFacade radioFacade;

    @GetMapping
    public String adminPage(Model model) {
        model.addAttribute("stations", radioFacade.getAllStations());
        model.addAttribute("playlists", radioFacade.getAllPlaylists());
        model.addAttribute("tracks", radioFacade.getAllTracks());
        return "admin";
    }

    @PostMapping("/tracks/add")
    public String addTrack(@RequestParam String title,
                           @RequestParam String artist,
                           @RequestParam("file") MultipartFile file) {
        radioFacade.uploadTrack(title, artist, file);
        return "redirect:/admin";
    }

    @PostMapping("/tracks/delete/{id}")
    public String deleteTrack(@PathVariable Long id) {
        radioFacade.deleteTrack(id);

        return "redirect:/admin";
    }

    @PostMapping("/playlists/add")
    public String addPlaylist(@RequestParam String name,
                              @RequestParam(required = false) List<Long>
trackIds) {
        radioFacade.createPlaylist(name, trackIds);
        return "redirect:/admin";
    }

    @PostMapping("/playlists/delete/{id}")
    public String deletePlaylist(@PathVariable Long id) {
        radioFacade.deletePlaylist(id);
        return "redirect:/admin";
    }

    @PostMapping("/stations/add")
    public String addStation(@RequestParam String name,
                             @RequestParam int bitrate,
                             @RequestParam(required = false) List<Long>
playlistIds) {
        radioFacade.createStation(name, bitrate, playlistIds);
        return "redirect:/admin";
    }

    @PostMapping("/stations/delete/{id}")
    public String deleteStation(@PathVariable Long id) {
        radioFacade.deleteStation(id);
        return "redirect:/admin";
    }

    @GetMapping("/playlists/edit/{id}")
    public String editPlaylistPage(@PathVariable Long id, Model model) {
```

```
        Playlist playlist = radioFacade.getPlaylistById(id);

        model.addAttribute("playlist", playlist);
        model.addAttribute("allTracks", radioFacade.getAllTracks());

        return "edit_playlist";
    }

    @PostMapping("/playlists/update")
    public String updatePlaylist(
            @RequestParam Long id,
            @RequestParam String name,
            @RequestParam(required = false) List<Long> trackIds
    ) {
        radioFacade.updatePlaylist(id, name, trackIds);
        return "redirect:/admin";
    }

    @GetMapping("/statistics")
    public String statisticsPage(Model model) {
        model.addAttribute("listenLogs", radioFacade.getRecentListenLogs());
        model.addAttribute("connectionLogs",
radioFacade.getRecentConnectionLogs());
        return "admin_statistics";
    }
}
```

Клас WebPageController забезпечує відображення головної сторінки та логіку авторизації.

```
@Controller
public class WebPageController {

    @Autowired private StationRepository stationRepository;
    @Autowired private UserRepository userRepository;
    @Autowired private PasswordEncoder passwordEncoder;
    @Autowired private RadioFacade radioFacade;

    @GetMapping("/")
    public String index(Model model) {
        model.addAttribute("stations", stationRepository.findAll());

        Authentication auth =
SecurityContextHolder.getContext().getAuthentication();

        if (auth != null && auth.isAuthenticated() &&
!auth.getName().equals("anonymousUser")) {
            model.addAttribute("isLoggedIn", true);
            model.addAttribute("username", auth.getName());
            boolean isAdmin = auth.getAuthorities().stream()
                    .anyMatch(a -> a.getAuthority().equals("ROLE_ADMIN"));
            model.addAttribute("isAdmin", isAdmin);
        }

        return "index";
    }

    @GetMapping("/login")
    public String login() {
        return "login";
    }
```

```
    @GetMapping("/register")
    public String register() {
        return "register";
    }

    @PostMapping("/register")
    public String registerUser(@ModelAttribute User user) {
        if (userRepository.findByUsername(user.getUsername()).isPresent()) {
            return "redirect:/register?error";
        }
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        user.setRole("ROLE_USER");
        userRepository.save(user);
        return "redirect:/login";
    }

    @GetMapping("/favorites")
    public String favorites(Model model, Authentication authentication) {
        if (authentication == null || !authentication.isAuthenticated() ||
"anonymousUser".equals(authentication.getName())) {
            return "redirect:/login";
        }

        String username = authentication.getName();

        model.addAttribute("username", username);
        model.addAttribute("isLoggedIn", true);

        boolean isAdmin = authentication.getAuthorities().stream()
                .anyMatch(a -> a.getAuthority().equals("ROLE_ADMIN"));
        model.addAttribute("isAdmin", isAdmin);

        var stations = radioFacade.getUserFavoriteStations(username);

        model.addAttribute("favoriteStations", stations);

        return "favorites";
    }
}
```

Діаграма класів

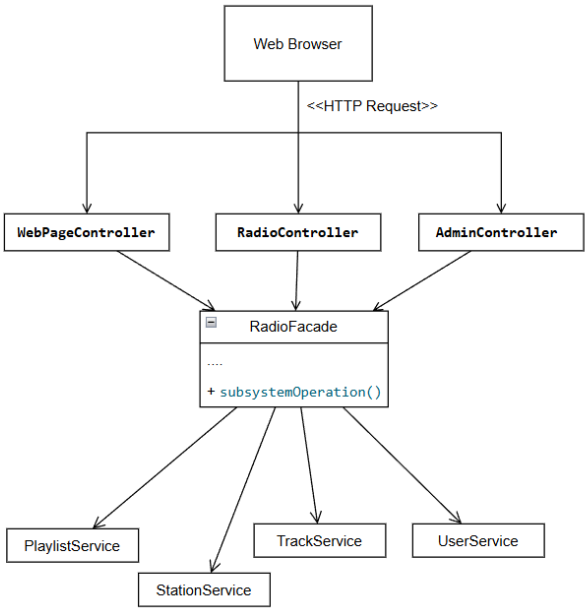

Рис. 1 Діаграма класів

Представлена діаграма ілюструє архітектуру клієнт-серверної взаємодії в розробленій системі, де веб-браузер виступає в ролі клієнта, що ініціює HTTP-запити до відповідних контролерів серверу. Ключовим архітектурним елементом є клас RadioFacade, який реалізує патерн Фасад для інкапсуляції складної бізнес-логіки та делегування завдань сервісним підсистемам, що забезпечує слабку зв'язність компонентів і дозволяє контролерам працювати з уніфікованим інтерфейсом без прямої залежності від внутрішньої реалізації сервісів.

Питання до лабораторної роботи:

1. Що таке клієнт-серверна архітектура?

Це найпростіший варіант розподілених додатків, що складається з клієнтів, які представляють додаток користувачеві та серверів, які використовуються для зберігання й обробки даних

2. Розкажіть про сервіс-орієнтовану архітектуру.

Це модульний підхід до розробки ПЗ, який базується на використанні розподілених, слабо пов’язаних сервісів зі стандартизованими інтерфейсами, що взаємодіють за стандартизованими протоколами.

3. Якими принципами керується SOA?

Принципи SOA включають використання розподілених сервісів, слабку пов'язаність, наявність стандартизованих інтерфейсів та взаємодію через обмін повідомленнями для надання певних бізнес-функцій.

4. Як між собою взаємодіють сервіси в SOA?

Взаємодія відбувається виключно шляхом обміну повідомленнями, зазвичай по протоколу HTTP з використанням SOAP або REST, без створення спеціальних інтеграцій для прямого доступу до спільних даних.

5. Як розробники взнають про існуючі сервіси і як робити до них запити?

Вони дізнаються про сервіси завдяки їх реєстрації на спеціальних сервісах реєстрах, а взаємодія часто покладається на централізований компонент – шину даних.

6. У чому полягають переваги та недоліки клієнт-серверної моделі?

Перевагою тонких клієнтів є простота розгортання та оновлення, а товстих – менші вимоги до сервера та можливість роботи офлайн; недоліком є залежність від доступності сервера та складність оновлення клієнтського ПЗ

7. У чому полягають переваги та недоліки однорангової моделі взаємодії?

Перевагами є децентралізація, стійкість до збоїв, рівноправність вузлів та розподіл ресурсів; недоліками – проблеми з безпекою, синхронізацією даних, складність контролю та зниження ефективності пошуку зі збільшенням мережі

8. Що таке мікро-сервісна архітектура?

Це стиль розробки, де додаток створюється як набір малих, автономних служб (мікросервісів) з чітко визначеними межами, що орієнтовані на конкретні бізнес-можливості та розгортаються незалежно

9. Які протоколи використовуються для обміну даними в мікросервісній

архітектурі?

Для обміну даними використовуються протоколи HTTP/HTTPS, WebSockets, AMQP та зв'язок на основі повідомлень

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли ми в

проєкті між шаром веб-контролерів та шаром доступу до даних реалізуємо шар

бізнес-логіки у вигляді сервісів?

Ні, такий підхід не є SOA, оскільки SOA передбачає використання розподілених сервісів, що взаємодіють через мережеві протоколи як альтернативу монолітній архітектурі, а не просто внутрішній поділ коду на шари.

Висновок:

У ході виконання лабораторної роботи було спроєктовано та реалізовано розподілену програмну систему для онлайн-радіостанції на базі клієнт-серверної архітектури з використанням технології Spring Boot та протоколу HTTP.